Rust Book - 3.3: Functions

#rust How Functions Work

> Rust code uses *snake case* as the conventional style for function and variable names - all letters are lowercase and underscores separate words.

> Rust doesn't care where (and what order) you define your functions, only that they're defined somewhere.

Function Bodies Contain Statements and Expressions

Function Bodies Contain Statements and Expressions

> Function bodies are made up of a series of statements optionally ending in an expression. Rust is an expression-based language, this is an **important distinction** to understand. Other languages don't have the same distinctions…

Statement & Expressions

> *Statements* are instructions that perform some action and do not return a value.

> *Expressions* evaluate to a resulting value.

> The `let y = 6` statement does not return a value, so there isn't anything for `x` to bind to. This is **different from what happens in other languages**, such as C and Ruby, where the assignment returns the value of the assignment. In those languages, you can write `x = y = 6` and have both `x` and `y` have the value `6`; that is not the case in Rust.

Expressions

Expressions evaluate to something and make up most of the rest of the code that you'll write in Rust.

> Expressions can be part of statements: in Listing 3-1, the `6` in the statement `let y = 6;` is an expression that evaluates to the value `6`.

> Calling a function is an expression. Calling a macro is an expression. The block that we use to create new scopes, `{}`, is an expression, for example:

```
fn main() {
    let x = 5;
    let y = {
        let x = 3;
        x + 1
    };
}
```

```
The value of y is: 4
```

This expression:

```
{
    let x = 3;
    x + 1
}
```

is a block that, in this case, evaluates to `4`. That value gets bound to `y` as part of the `let` statement. Note the `x + 1` line without a semicolon at the end, which is unlike most of the lines you've seen so far. Expressions do not include ending semicolons.

If you add a semicolon to the end of an expression, you turn it into a statement, which will then not return a value…

Functions with Return Values

Functions with Return Values

Functions can return values to the code that calls them. We don't name return values, but we do declare their type after an arrow (➡). In Rust, the return value of the function is synonymous with the value of the final expression in the block of the body of a function. You can return early from a function by using the `return` keyword and specifying a value, but most functions return the last expression implicitly.